Lecture 2

History of modern scientific computing

Literature (optional)

1. Paul E. Ceruzzi “A history of modern computing”, 2nd ed., MIT Press 2003
2. http://computerhistory.org - Computer History Museum online.

Supercomputing today

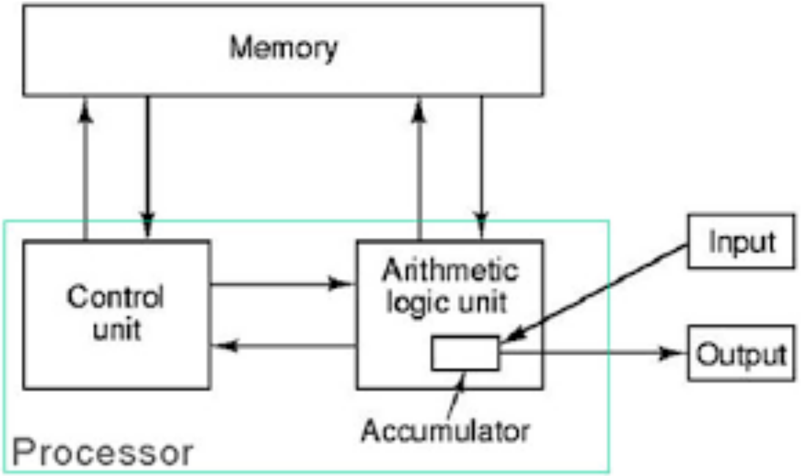
Python3 – basic concepts and grammar

Literature (required)

Textbook 1, Izaac and Wang, Computational QM

PSCB57. Intro to Scientific Computing.

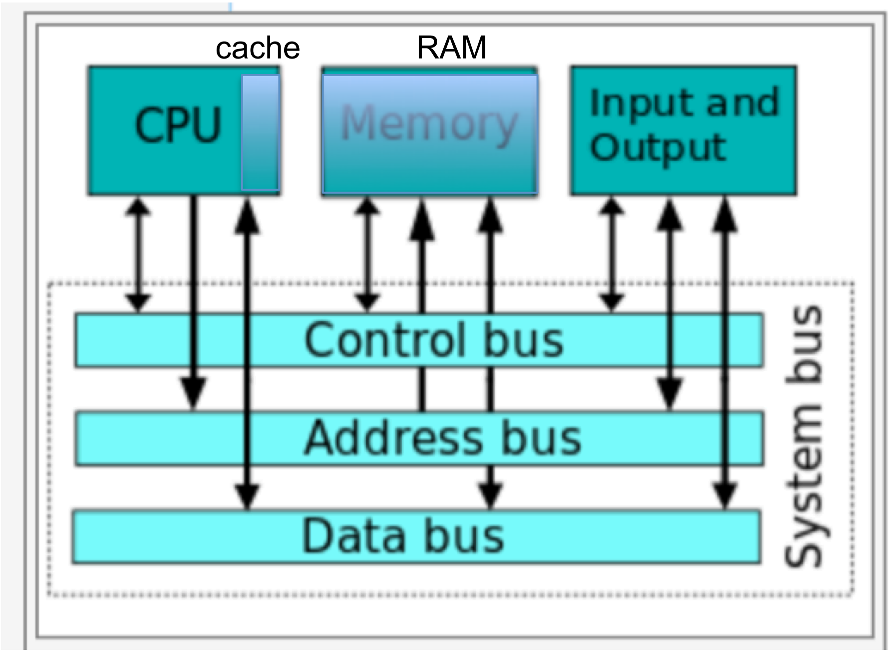
Von Neumann computer implements the logic of *Turing machine* – a model, where data and program are stored in the same memory,



CPU

Alan Turing
1912–1954

John von Neumann
1903–1957



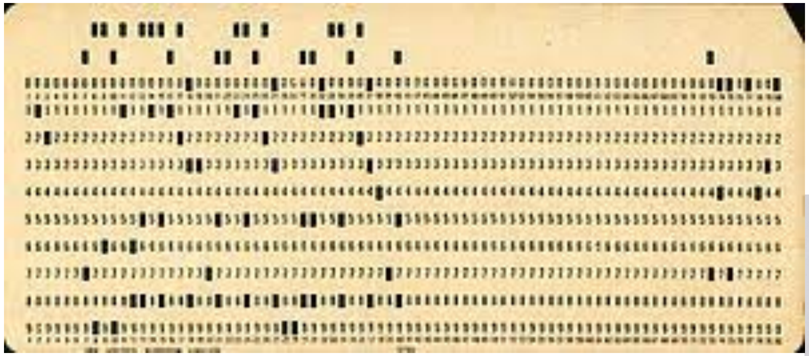
modern computer architecture

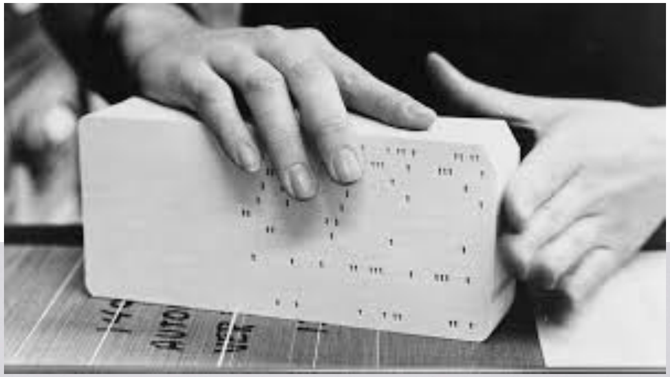
Memory = Random Access Memory (volatile, transistors need small power to represent 0s & 1s)

Storage = Nonvolatile Memory
(hard disk, solid state mem., earlier:
magnetic tape, punched paper) work like I/O

Mid-1960s and 1970s

IBM (International Business Machines Co.) grew when it transferred from decks of punched cards (very popular among businesses)

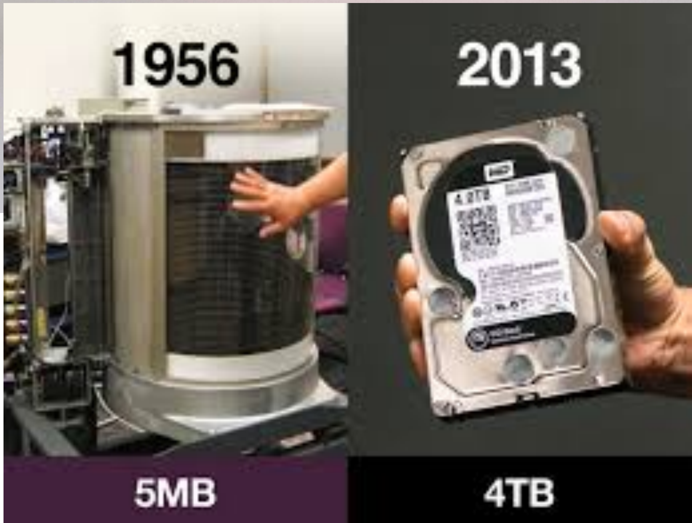




... and perforated tape storing programs

to magnetic media: tape, drums, disks

Q: How many times more data is stored on a 2013 hard-disk than on 1956 version?



Mid-1960s to 1970s

IBM captured 70% of global computer sales, mostly with very expensive ***mainframes*** (or supercomputers, bought by big firms, government centers and the universities). The rest of the market was served by Univac, Computer Data Corp., Sci. Data Systems, and Cray, which tried to emulate IBMs scale, salesforce and success.

DEC = Digital Equipment Corporation appeared as a small company in 1960s, located in old Abbotts Mills near Boston, benefitting from MIT connections. DEC, later called **Digital**, started a quiet revolution in the market by offering:

- much smaller ***minicomputers***, costing much much less than mainframes
- For instance, model PDP-8 costed $18k instead of IBM/360 for $2 million.
- Similar speed on some tasks, while much less data storage capability etc.
- Circuits in 1960s were non-integrated (cf. below) but fast-switching transistor-based, which allowed fairly large computational speeds
- Innovation in comp. architecture: *the interrupts* in CPU operation to serve input/output requests from the peripheral devices (printers, storage, keyboards, monitors)
- Different & much less pretentious culture in workplace and in sales
- OEMs (Orig. Equip. Manufacturers) appeared when Digital opened up their architecture, encouraged modifications

1970s and later...

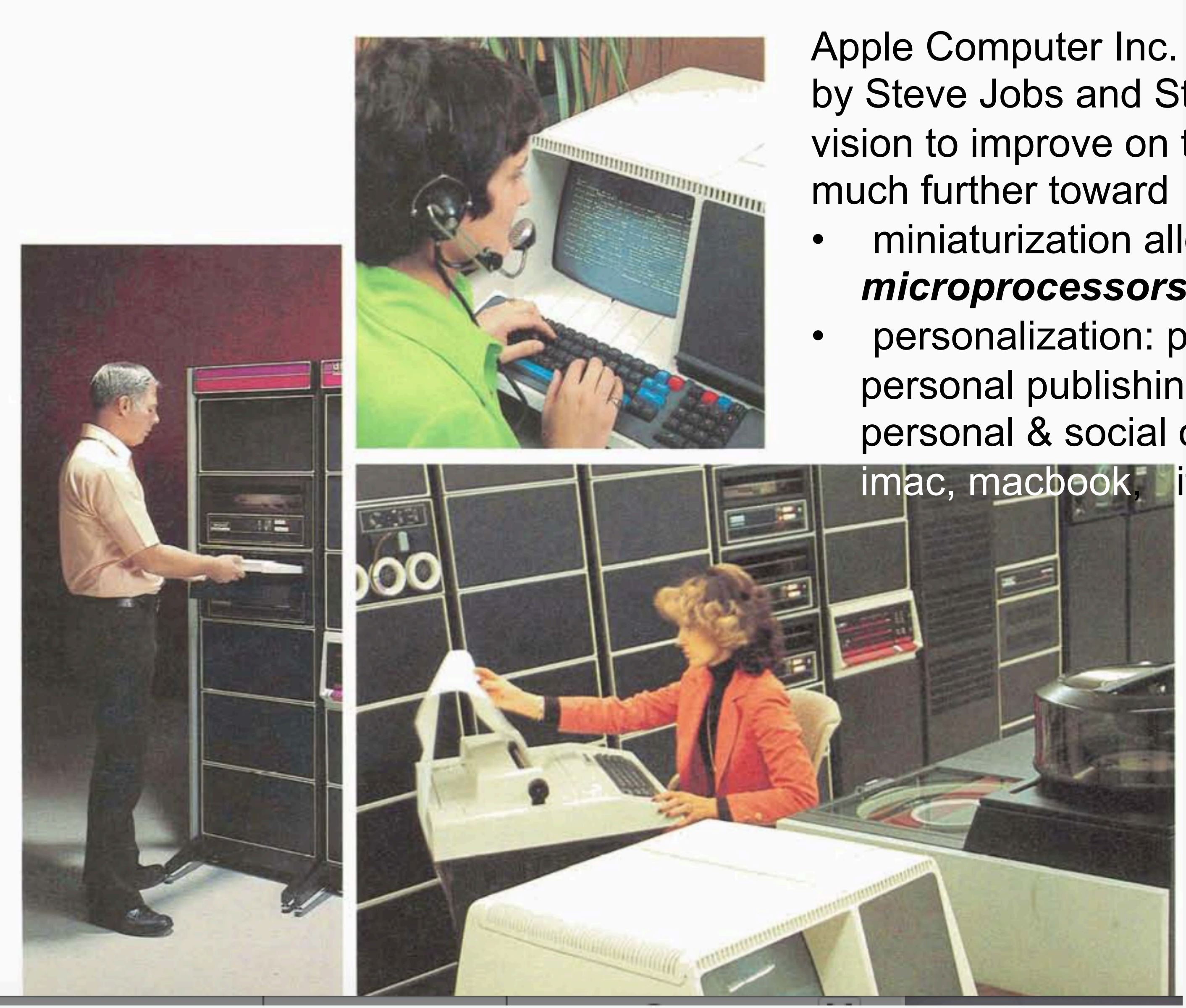
- **PDP-8** (left)
- very little memory
- but fast and cheap

- **PDP-11** (right)
- 128k RAM
- 2 MB removable hard disk
- > 30 kFLOPS arithmetic speed
- interactive consoles, time-sharing operating system, as opposed to batch input on mainframes

- Languages such as ALGOL, Pascal, and other became popular but later faded away. IBM’s PL/I (Programming Lang. I) was a commercial failure.
- **DEC** offered the first **UNIX operating systems** on PDP-line computers
- **C** appeared in 1973 and later **C++**, high-level languages that unlike numerical computation-oriented Fortran for “number crunching”, were not meant for scientific simulations. They were great for writing operating systems (fully C-based Unix in 1973, later **Linux** in 1990s, which is a modified Unix), and for interfacing at low level with hardware (low-level here means using elementary, simple instructions).

THE C PROGRAMMING LANGUAGE

Interactive use of Digital’s PDP-11 minicomputers (here shown in year 1977) was a harbinger of and a model for personal computers invented right then and getting huge popular in 1980s.



Apple Computer Inc. was founded in 1976 by Steve Jobs and Steve Wozniak, with a vision to improve on that achievement & go much further toward

- miniaturization allowed by small ***microprocessors***, and
- personalization: personal computing, personal publishing and all other things personal & social on Apple, Mac, ibook, imac, macbook, itunes, iphone, ipad, ...

late 1970s to 1990s

- **programmable calculator and PC revolution** based on *semiconductor technology in microchips* or *microprocessors.* Such *integrated circuits (IC) measure up to several cm* & house many solid-state transistors, as well as resistors and capacitors.

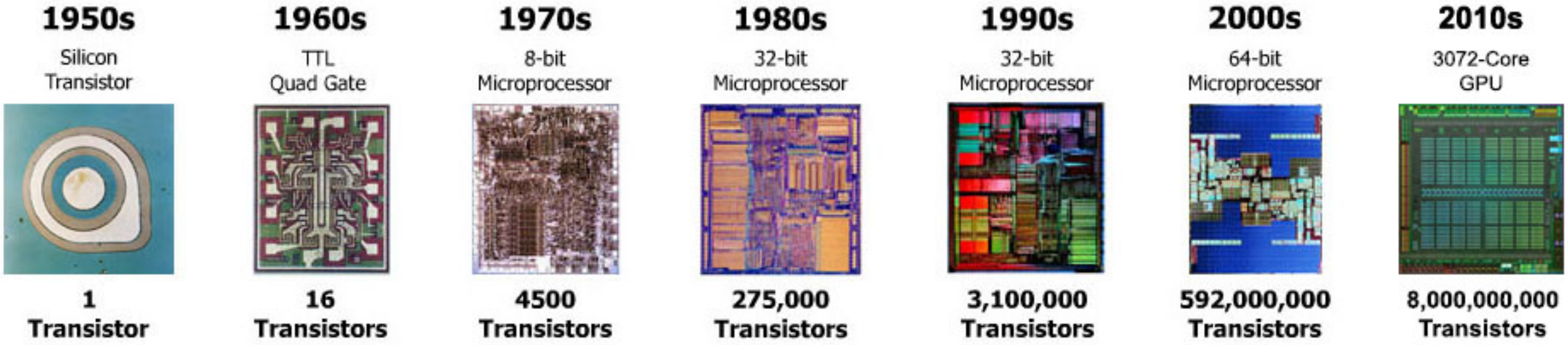
- In 1971-1974, **Intel** (4004; 8008, 8-bit) and **Motorola** (8-bit 6800)
- Intel 8086 was a 16-bit CPU (central processing unit) from 1978, → "x86_16"
- Motorola 68000 was a 16/32-bit CPU from 1979

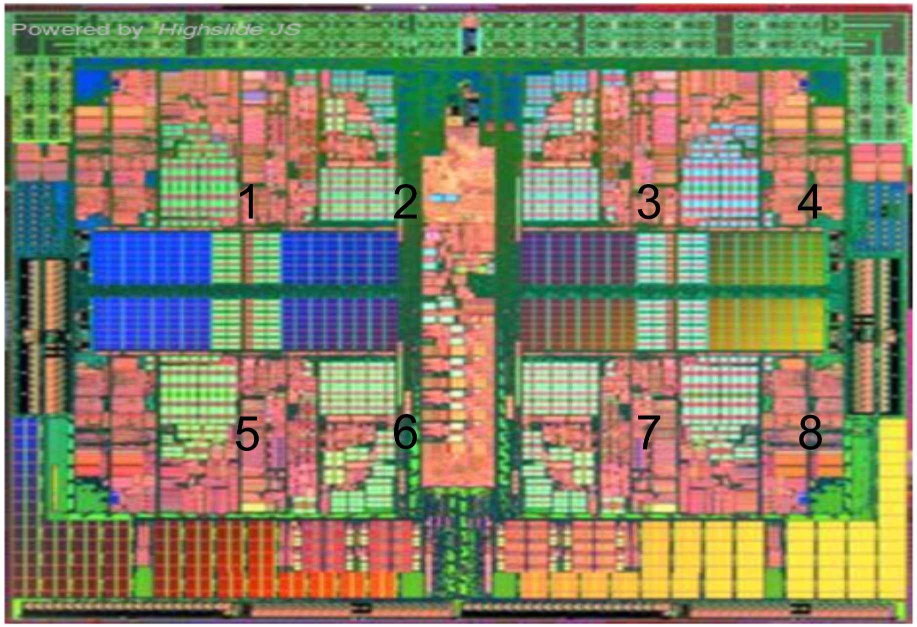
- In 1980s and 1990s computers spread outside academia, big business and military. Microcomputers or desktop Personal Computers (PCs) moved into every house and school, at the end of 20th century into the backpacks as laptops, and finally into our pockets as smartphones, not for scientific computation but rather for a simpler task(?) of communication (networking). This interesting history is outside the scope of our course, even though for your computing will likely choose a laptop

- Simplified languages like BASIC, and in late 1970s/early 1980s the scripting languages MATLAB, IDL have appeared. They are *interpreted*, not compiled, which has its good and bad sides.

- **Python** was created in 1989 and became popular in 1990s and 2000s.
- Creator, Guido van Rossum, was named the Benevolent Dictator for Life. Now the project is guided by Steering Council. We are going to learn Python3 version (2008).

1970s—2010s, the integrated circuit revolution
as a basis of all our current technical civilization: 40+ years of **Moore's law**

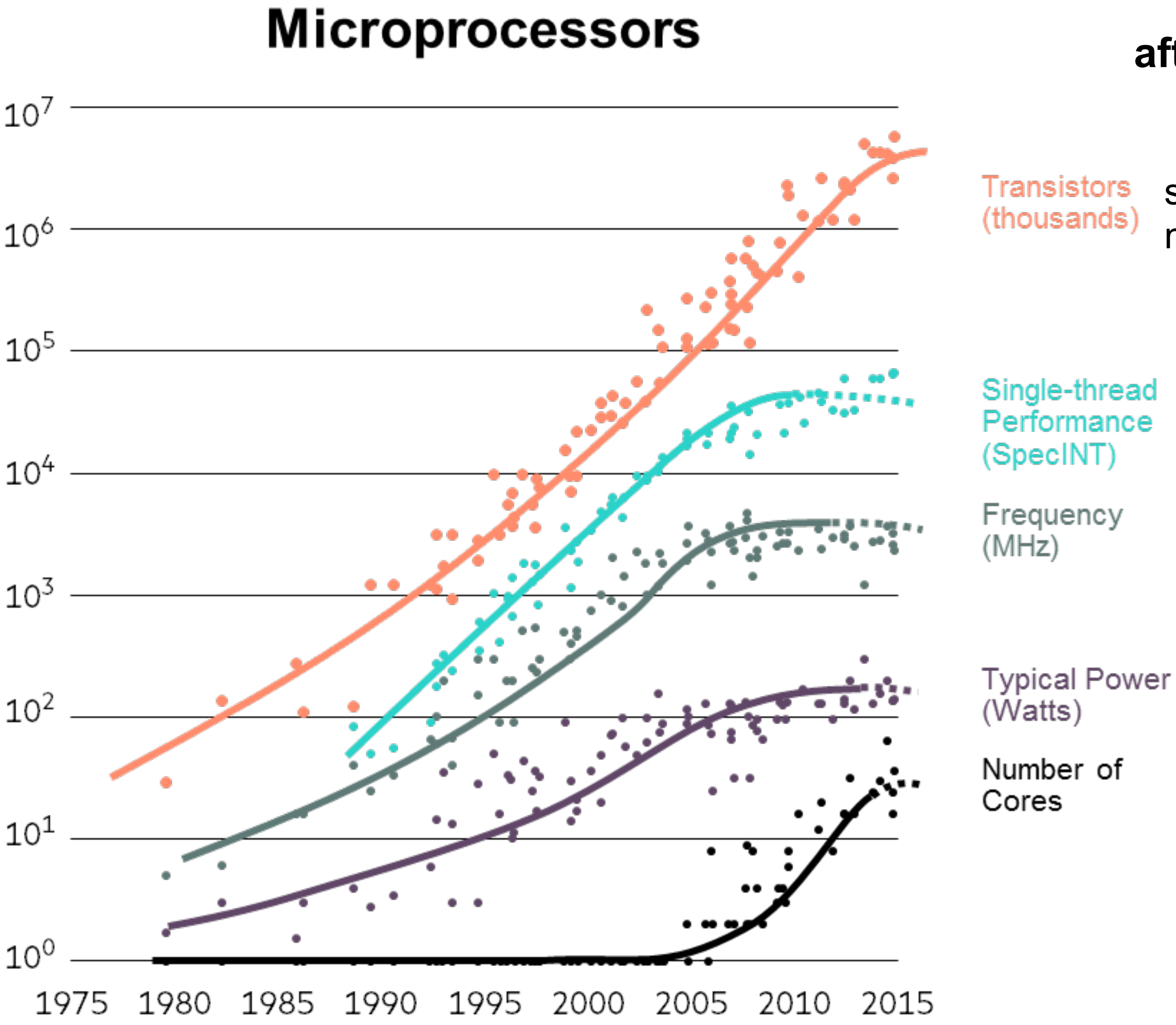




A highly integrated circuit has both arithmetic+logic units (ALU), fast cache memory, and communication circuits, all in a small package of Central Processing Unit (CPU).

N ~ 10^{10} transistors

This processor looks like a CPU or part of a GPU (Central or Graphics Processing Unit).
It is not easy to execute one calculation simultaneously on many **cores (= sub-CPUs)** of a CPU.
We call such processors **multicore**, and programs running on all cores **multithreaded**.



after ~2004:

still goes up maybe slower

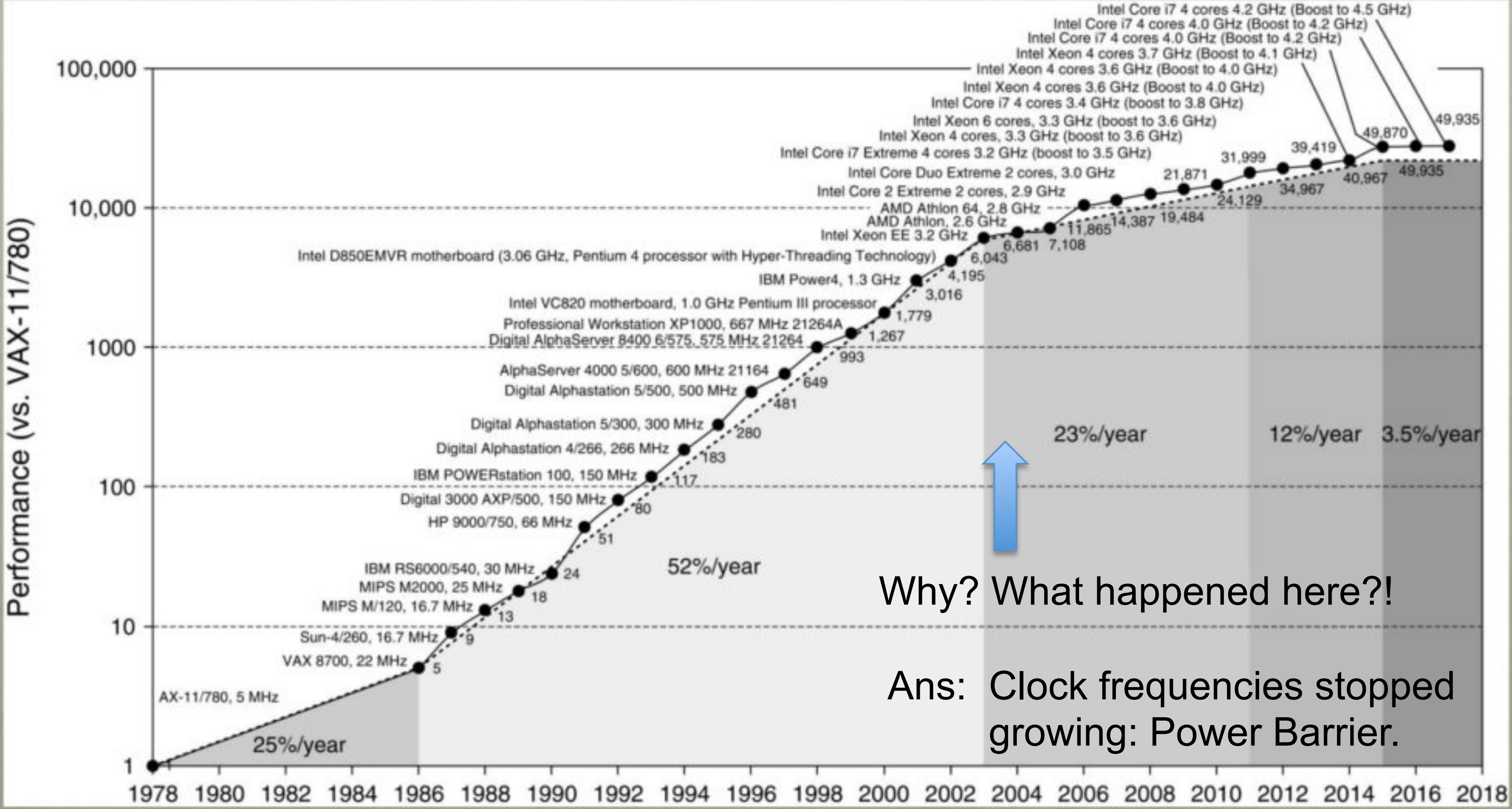
stagnated

stagnated

hit a wall

started climbing

Uniprocessor Performance (Single Core)

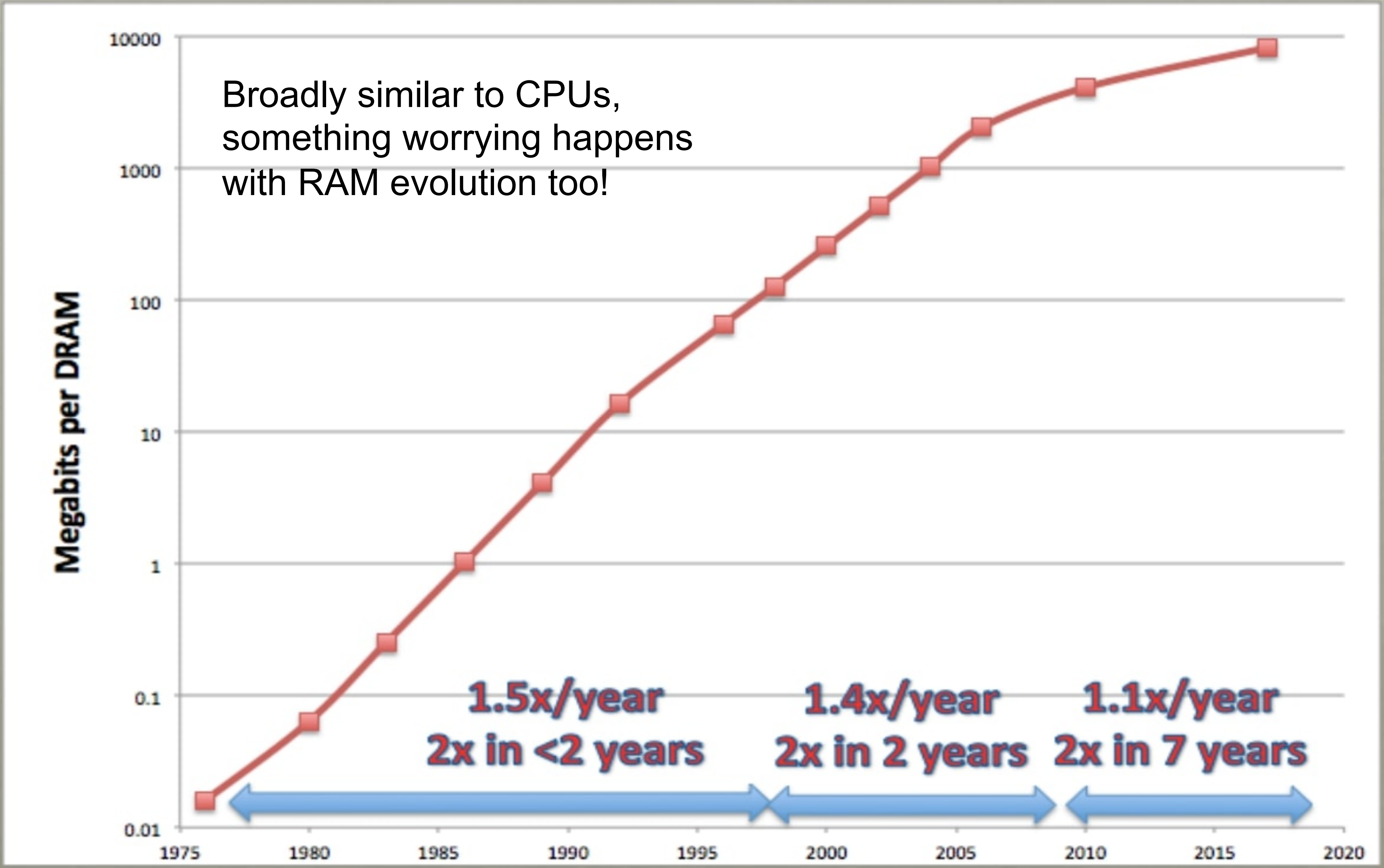


Why? What happened here?!

Ans: Clock frequencies stopped growing: Power Barrier.

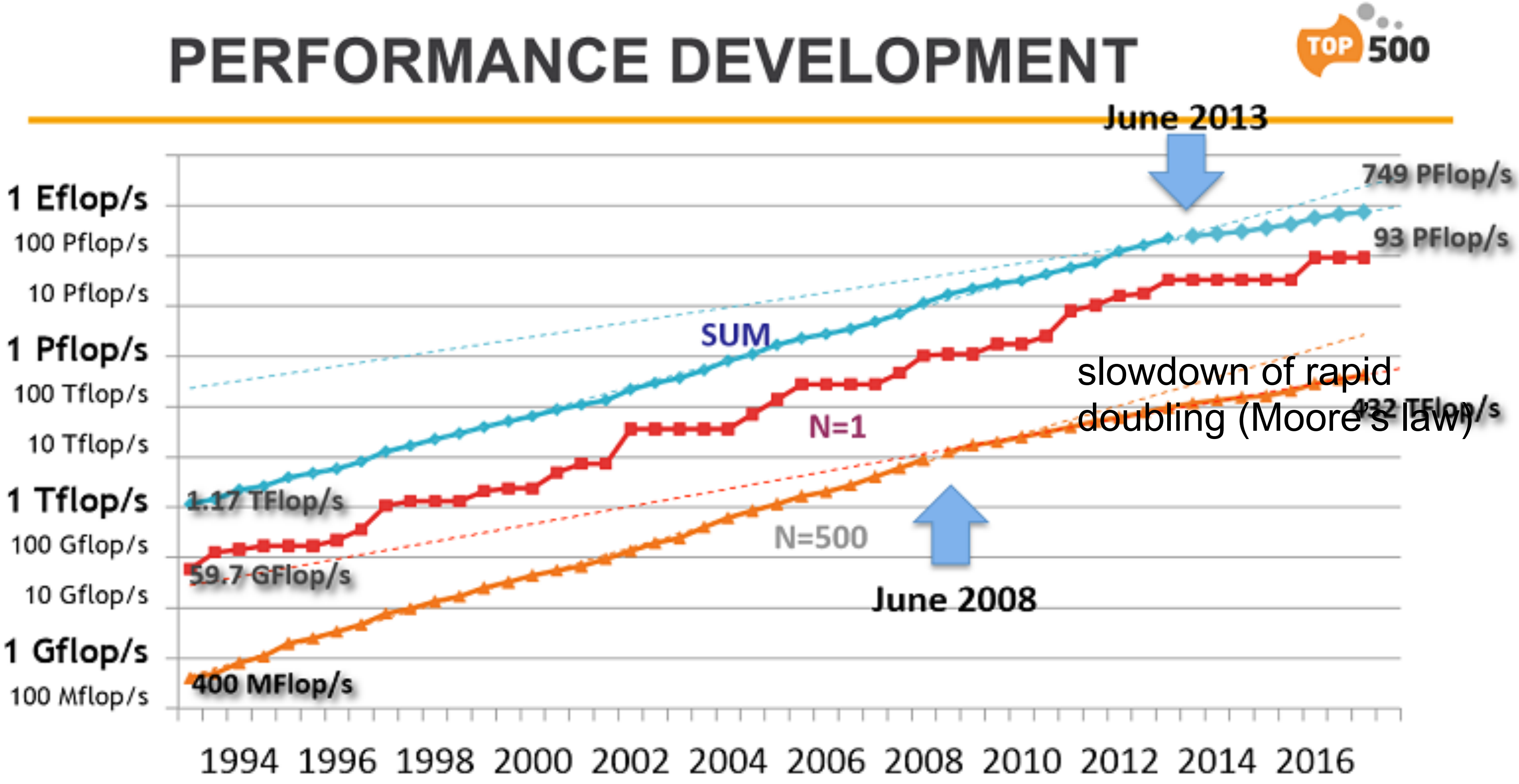
Moore's Law in DRAMs (memory)

https://www.nextbigfuture.com/2019/02/the-end-of-moores-law-in-detail-and-starting-a-new-golden-age.html = a DARPA talk



Broadly similar to CPUs, something worrying happens with RAM evolution too!

Top 500 supercomputers in the world since ~2008 grow in speed slower than before



slowdown of rapid doubling (Moore's law)

Moore's law says that the number of elements (logical gates, transistors) on an integrated circuit (say,on 1 cm^2 od a chip), grows exponentially with time, such that it doubles every ~2 years.

This was not a coincidence, but a combination of:

(i) repeated shrinkage cycles of circuitry etched onto silicon wafers, &

(ii) physics of materials, which after the shrinkage allowed to keep the *same amount* of electric power to the larger number of logical gates

Let's prove it using Physics.

A transistor on IC has some capacitance C and some stored charge Q.
It also has some resistance R on which current's energy can be dissipated.
Transistors discharge and re-charge, under voltage V.

The process of charging/discharging is not for free, rather it dissipates energy equal to $\Delta E = CV^2$. That's because $C=Q/V$, and energy drawn from voltage source V by charge Q is $QV = (CV)V = CV^2$. Usually all of that energy must be supplied but is wasted as heat during discharge through inevitable resistance R.

If the chip works with clock frequency f, then the rate of energy dissipation (power) is

$P \sim N\ C\ V^2\ f$

where N is the number of transistors.

Now let's **shrink** the circuitry on an IC. Each side of the chip is constant but the pattern of circuits is shrunk by a factor **√2 = sqrt(2)=1.4128..** on a side, as it was in fact done every 2 years or so for 4 or 5 decades.
Q: What is happening to quantities C, V, and f of a transistor, and their number N?

N grows by a factor 2
[since there are 1.41 more transistors along each side of the chip

C grows by a factor 1/sqrt(2)
[this follows from electrostatics; C ~ area of capacitor/spacing between electrodes]

V was being increased by a factor 1/sqrt(2)
[i.e. decreased by 1.41..]

f was increased by a factor sqrt(2)
[because a smaller transistor can switch its state proportionally faster]

We've said that power needed to do calculations scales like $P \sim N C V^2 f$,

so it grows by a factor of: 2 (1/sqrt(2)) $(1/\text{sqrt}(2))^2$ sqrt(2) **== 1 (!)**

We have shown why 15+ cycles of shrinkage of transistors & circuits on a microchip over several decades did not require much more cooling + much more power supply to the chip. (Electric power requirement grew because the IC's area tended to grow.)

The exponentially growing number N was utilized to complicate the logical structure of CPUs, e.g., by introducing more and more levels of *cache memory* inside the processor. This memory is much fast but much smaller than RAM. It can be used as a buffer between CPU and the outside operational memory (RAM).

This allowed to mask the fact that CPU-RAM communication and the speed of the RAM (memory) was growing slower than the processing needs of CPU. As a result, frequency f grew ~2 times, instead of just 1.41 in each new generation of processors.

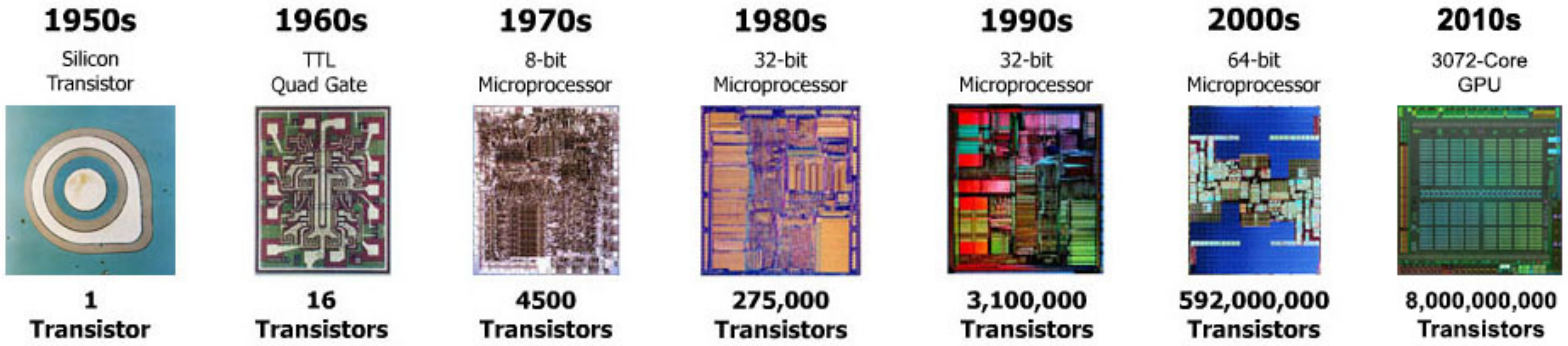
The progress *was* phenomenal.

If our cars were increasing in speed at the rate equal to the increases in computing power, then we would now drive at cosmic speeds close to the speed of light, or about once around the Earth per second.

No other area of human activity saw a million-, let alone a billion-fold quantitative improvement in one human generation, but computing did.

Physics allowed us to supply a similar power to a faster, smaller processor.

So why the size reduction and exponential growth of clock speed had to end around 2004?



Transistors inside the microchips, now counted in billions, became so small that, due to microscopic material imperfections, electric power gets dissipated in new, unavoidable way, via the *leakage current* in volatile memory (RAM, CPU).

Transistor should either block or transmit current, but now the blocked state became not 100% blocked. Electric energy is slowly going to waste, without carrying out any computation. That’s how the doped semiconductors work.
Leaks occurred before, but were masked by much larger heat dissipation due to (useful) toggling from state “0” to state “1”, and from “1” to “0”, during binary data processing.

We now live in a technical civilization where:
(i) size of circuitry still diminishes (with manufacturing problems, but diminishes)
(ii) number of transistors in new generations of processors still grows (slower)
(iii) f ~ const. ~3-4 GHz (clock speed of 1 core of CPU inches forward, but very slowly) in order to try to keep energy supply and dissipation at a reasonable level!

We call it the power barrier.

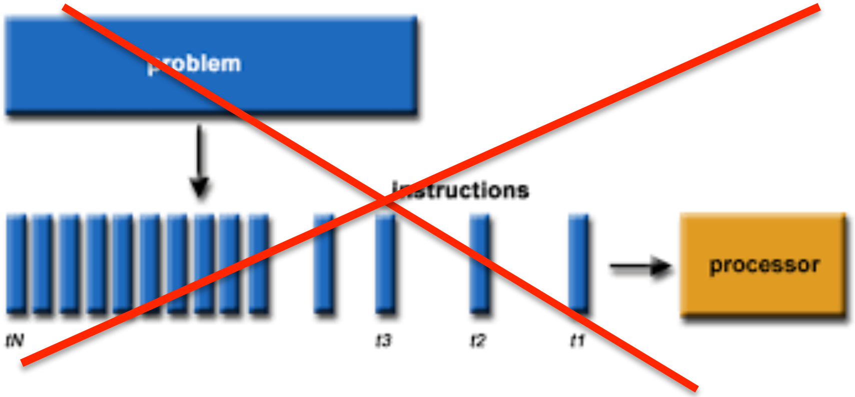
It is no longer possible to play the game of “increase clock speed and keep one core”, as before. On the contrary, increasing the number of cores in a processor comes cheap (power-wise), and therefore is now practiced by manufacturers.

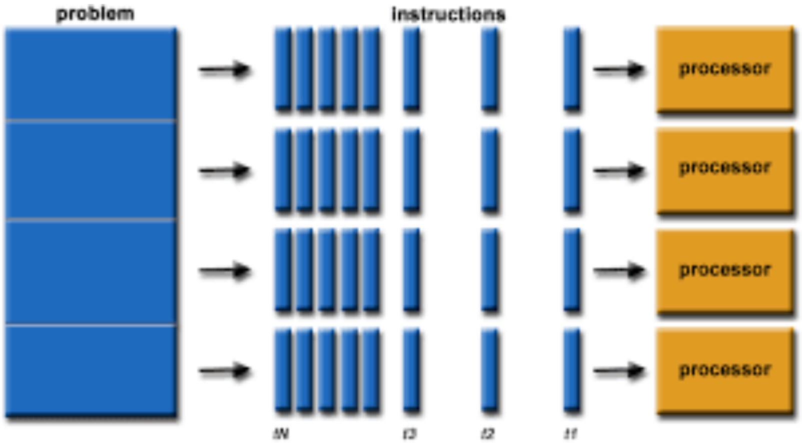
We are (and will in the near future) be getting processors with more and more cores. Each of these core will NOT be much more capable to crunch numbers than the previous hardware generation.

Thus the power barrier forces everyone to use more and more cores **in parallel**, if we want to compute a given problem faster.

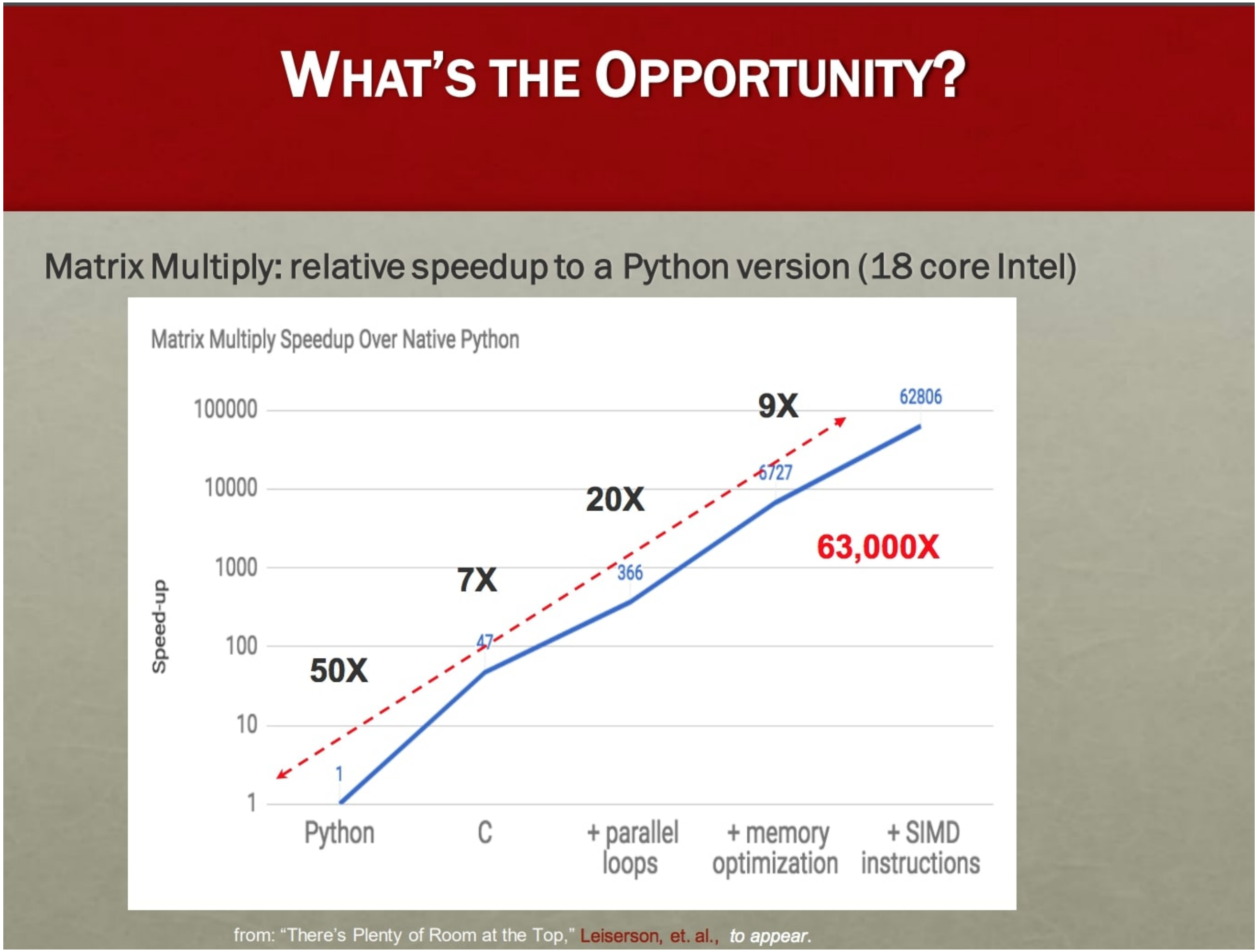
And this is more difficult in Python than e.g. in C or in Fortran!

(parallelization will be discussed in course PHYD57)





Parallel programming – is it worth it? Yes, but we need to crawl before we run. Courses: PSCB57 = single-threaded programming, PHYD57 = C/Fortran + deeper look at numerical methods + code optimization on modern hardware.

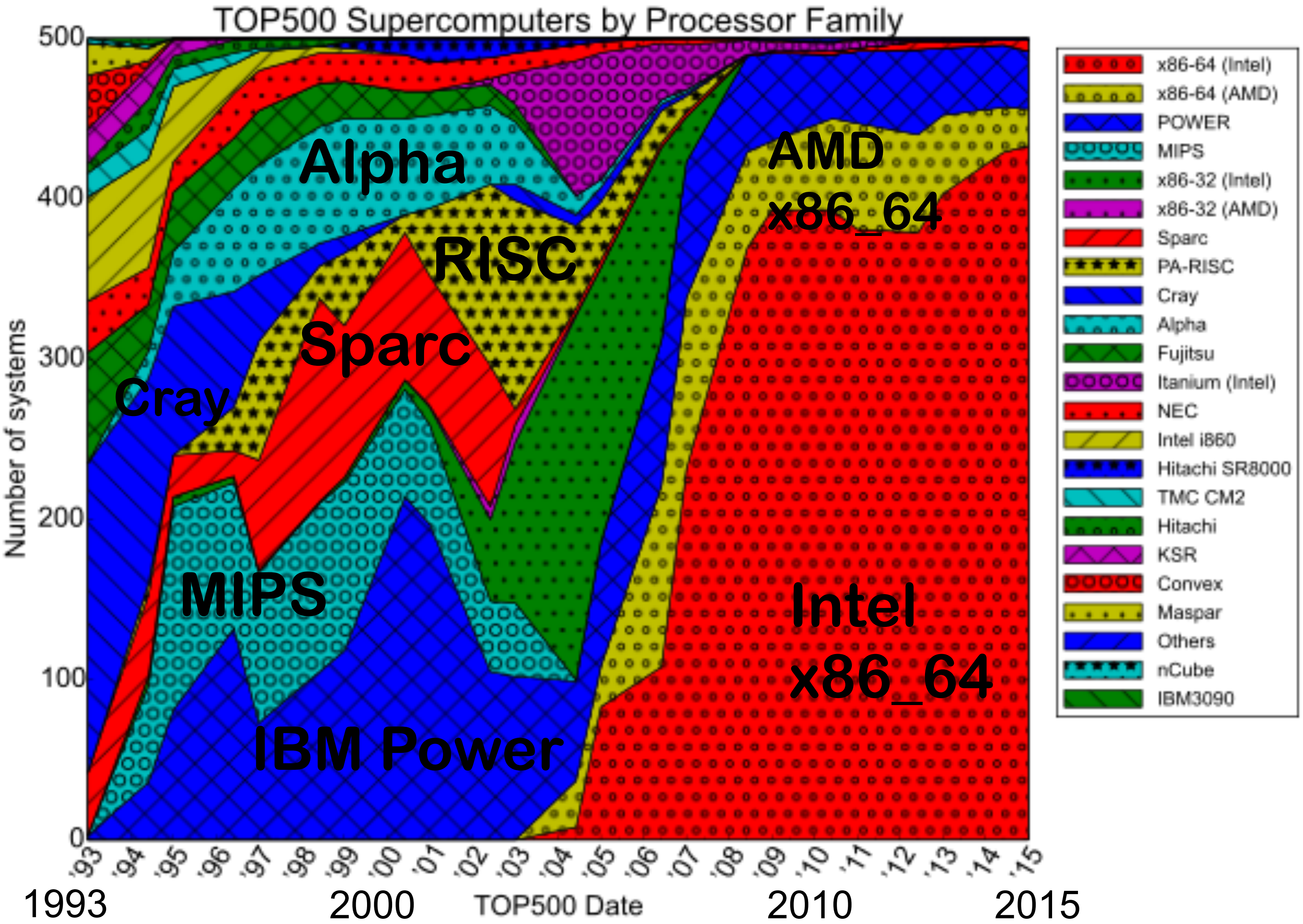


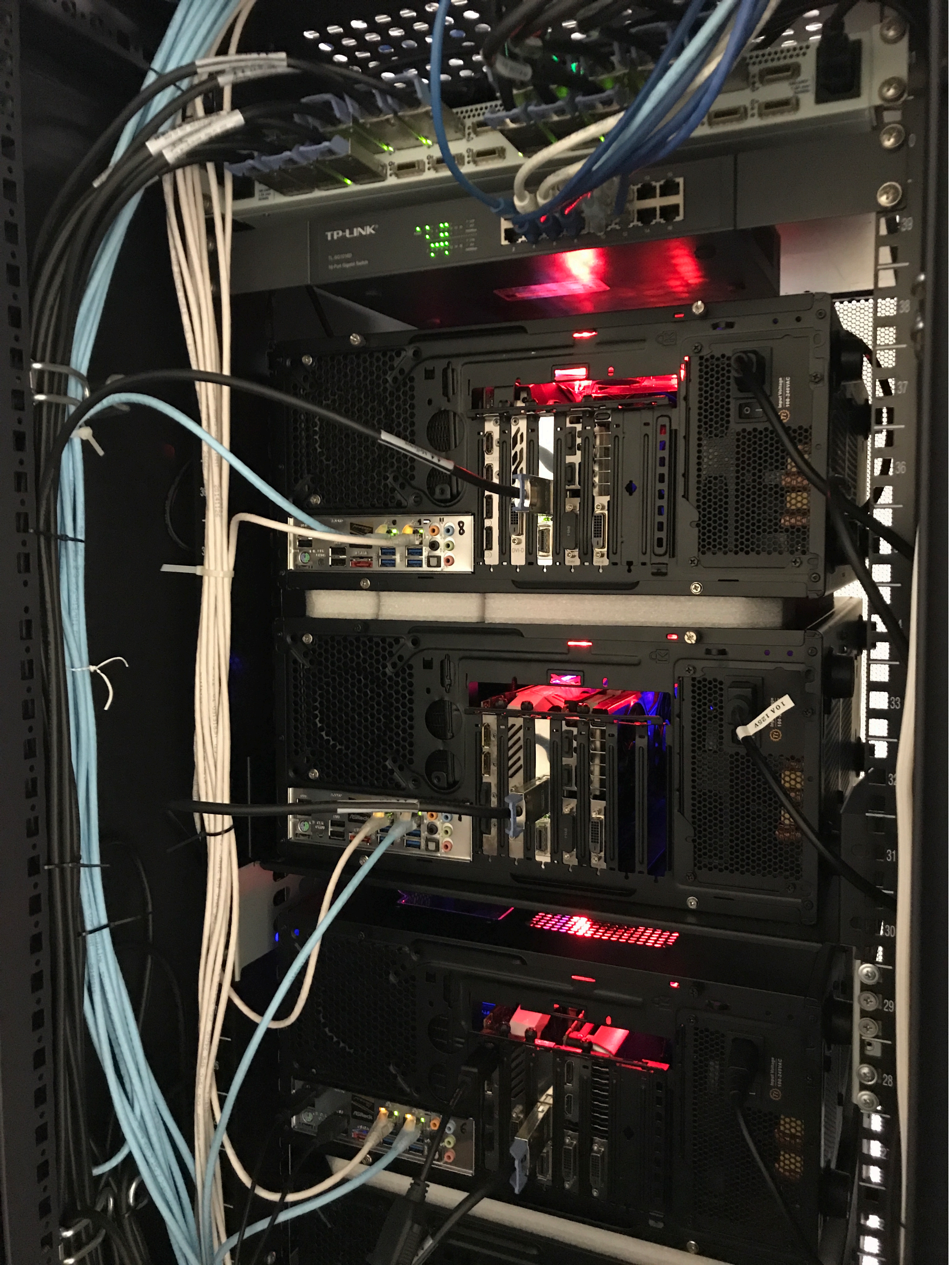
WHAT'S THE OPPORTUNITY?

Matrix Multiply: relative speedup to a Python version (18 core Intel)

from: "There's Plenty of Room at the Top," Leiserson, et. al., *to appear.*

Processor architecture & instruction set: Out of creative chaos of different CPUs before ~2004, Intel's x86_32/64 processors emerged as big winners. Check your laptop! Your CPU and CPUs in supercomputers are both x86_64 microprocessors, they understand the same instruction set.   Here is the full story, in 1993-2015:





Modern supercomputing clusters compute in parallel.

They consist of many nodes (workstations running Linux operating system), connected by a fast network in order to work in parallel on a single problem (or on many problems simultaneously). Hierarchical hardware inside as well.

Even though the modern networks (Ethernet, Infiniband) transfer from 1 to 20 GB/s between arbitrary two nodes using networking switches (at the top of the picture, there is 2 GB/s Infiniband switch), data can flow much faster, at 250-500 GB/s, inside the computational cards.

Node connectivity can therefore be a bottleneck for some (fortunately not all) parallel computer applications.

Picture shows some of the 28 nodes of the UTSC supercomputer designed and built in 2017 by prof. P. Artymowicz + u/g student.
Computations are done by:

- CPUs (6-core, 4GHz overclock),
- GPUs (graphics cards), and/or
- Intel Xeon Phi cards (57-core processors)

Each node contains CPU + 2 Nvidia graphics cards capable of combined ~10 TFLOPs in single precision.

This science cluster can compute 10^{12} (trillion) times faster than ENIAC in 1946, and

~6 billion times faster than similarly sized PDP-11 (1970), using
~10 million times larger memory (RAM, disks) than PDP-11/45.

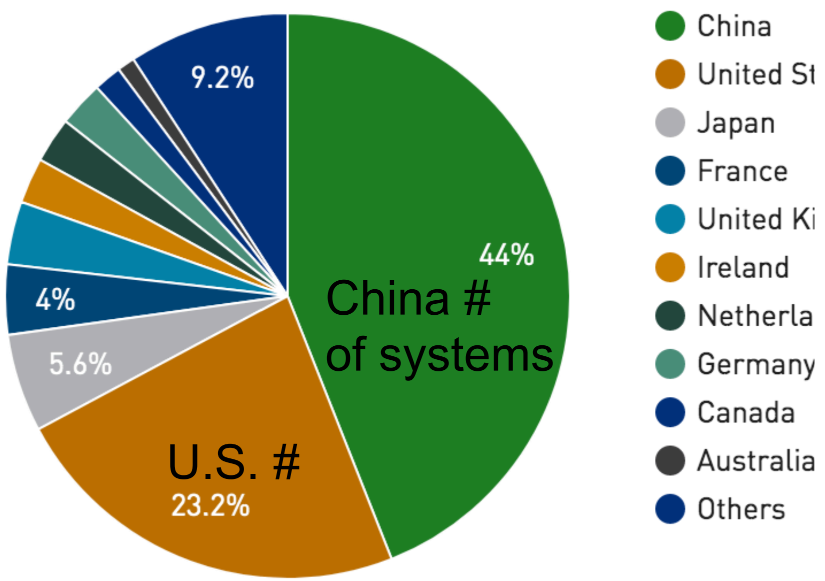
It costs ~10% of PDP-11.

Who has the fastest supercomputers today? US & China.

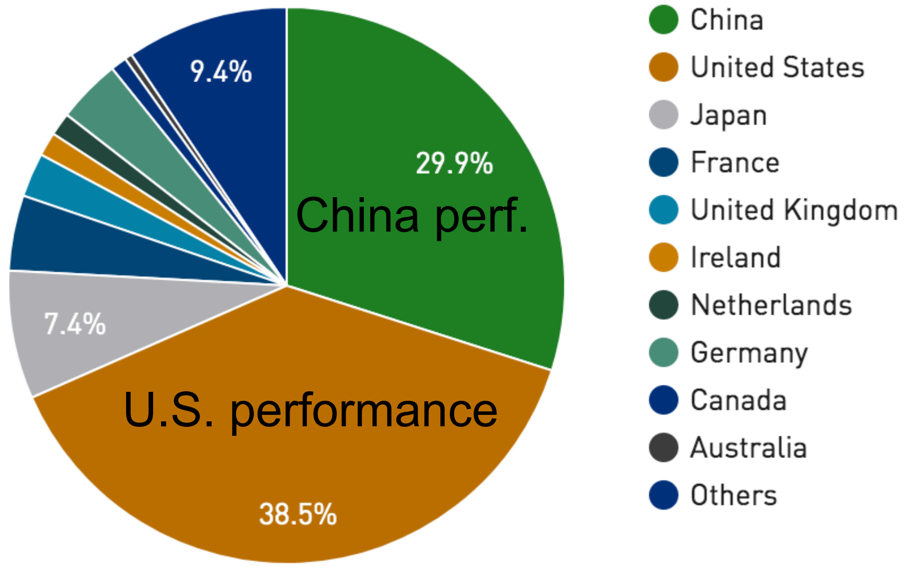
(U.S. used to be the dominant player for 65 years, 1945...2010)

2019

Countries System Share



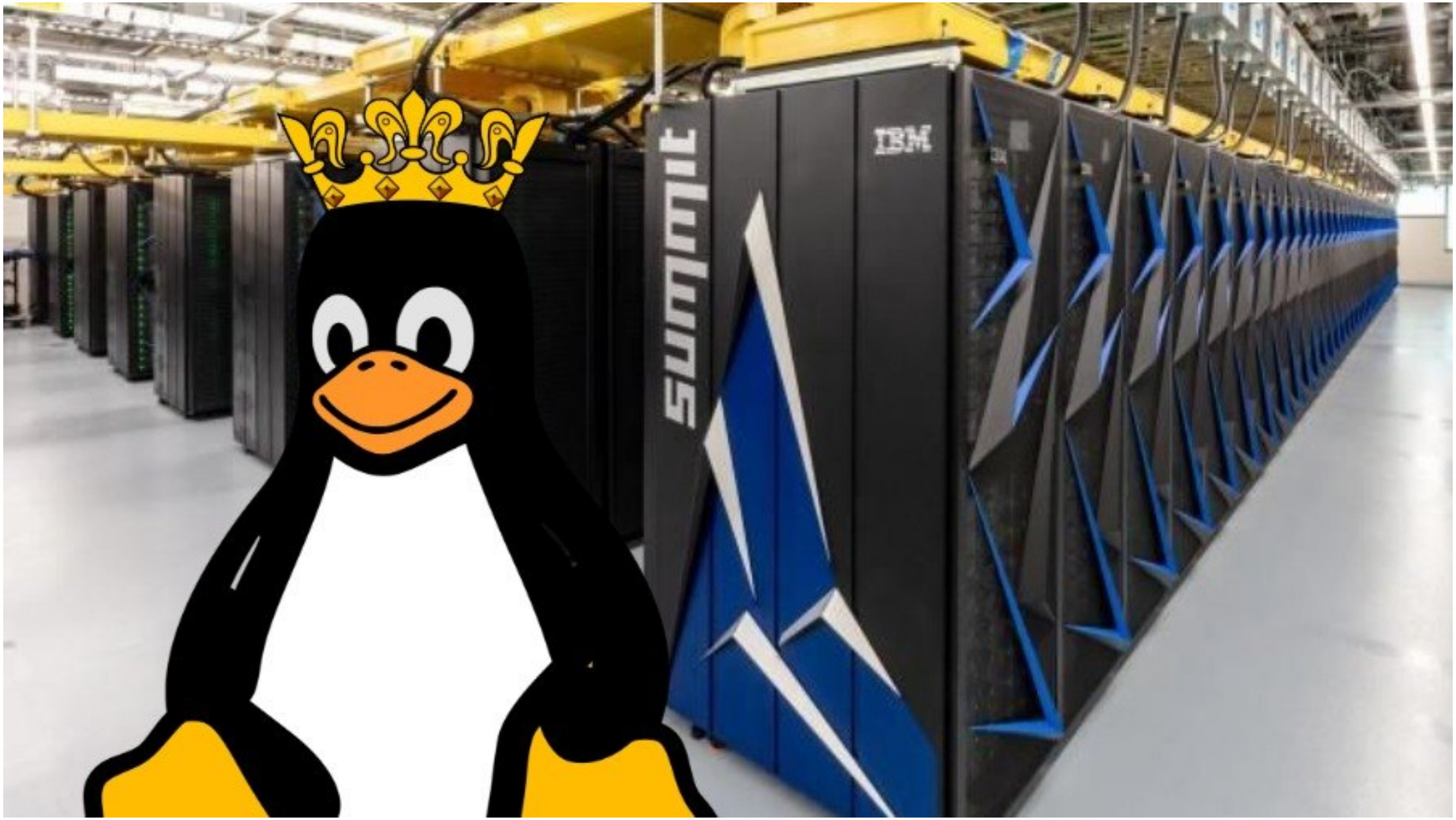
Countries Performance Share



Title of media report in November **2017**: “China overtakes U.S in the Top 500 Supercomputers List”

China also had two fastest supercomputers in 2017 (Sunway & Tianhe-2), including one based on own 240-core processors and one on Intel Xeon Phi processors (60-core). Up to date statistics available on top500.org

0.2% of top 500 systems were Chinese, 54% were American in 1999
4% --- ,, --- 55% --- ,, --- in 2009
44% --- ,, --- 29% --- ,, --- in 2019



Title of article in June **2018**:
"Linux Powers ALL Top 500 Supercomputers in the World. U.S. beats China for #1"

Computer called *Summit* in the U.S. is now the fastest machine in the world.
Linux succeeded Unix in its domination of HPC (High Performance Computing);
MacOS is Linux derivative, no role in HPC. Windows OS was never a viable choice.

Modern top supercomputers:
rows of 19”-wide racks filling basketball fields, 1000s of nodes (workstations), consume up to ~23 MW of electric power and emit heat at that rate.
Cost ~$300 million.
Not environment-friendly (e.g. UofTs SciNet is warming/damaging env., whose warming it was meant to study.)

However, let’s put things into perspective! That’s about the price of *ONE* Boeing 777 airplane, whose engines also produce 23 MW of power in cruise & much more while climbing.
There are 1600 B777s, and 10000 Airbuses!

Summit has 4608 nodes, 9216 IBM POWER9 CPUs and 27648 Nvidia Tesla GPUs.
Most computational power is from those Tesla V100 graphics cards.

The combined performance is measured at ~150 PFLOPS, out of the theoretical number of 200 PFLOPS (Summit can perform 150,000,000,000,000,000 arithmetic ops/s).

Modern top supercomputer:

150,000,000,000,000,000 arithmetic operations/second.
Why so fast? Who’d ever need that?

To begin with, no single user is allowed to hog the whole supercomputer, there are thousands of users at national supercomputing centers.

We will do a realistic estimate of how long one scientific model runs on Summit.
[Do not memorize it! It’s informative, but not exact to better than on order of magnitude.]

Suppose 1000 scientists want to do high-resolution simulations in 3D, simultaneously. Each of them divides simulated object or region of space into, say, 2000x2000x2000 cells, or 8G cells [$(2K)^3$ = 8G]. Each cell must hold 5-10 floating point numbers of length 8 Bytes (double precision). For instance, in a fluid simulation those would be density and pressure, as well as 3 components of velocity vector in each cell. Storage of at least 40 to 80 B/cell is usually needed.

The total volume of simulated data may thus be ~1000* 8G*(40-80 B) ~ 500 TB.

Modern top supercomputer:

150,000,000,000,000,000 arithmetic ops/s. Is this not an overkill and a waste of money?!

If the total volume of simulated data is ~500 TB, and ~30K graphics cards do simultaneous number crunching on Summit, then

~500GB/30 ~16 GB

of this data must be processed by each GPU in each time step. (Today, 16 GB can fit on the biggest RAM available on GPUs, avoiding transfer from/to CPU RAM, a good thing!) Constantly shuffling data back and forth to RAM at a bandwidth of ~300 GB/s, the GPU card takes at least

~(16 GB) /(300 GB/s) = ~0.05 s

to process its allotted data in a given time step of the simulation (inter-processor and inter-node communication takes additional time, sometimes majority of time, but let's assume this does not apply to CFD = Computational Fluid Dynamics.)

Next, how many time steps are needed?

Modern supercomputers:

Large-scale simulations take a lot of time, so as to use all available resources! There is never "too much resolution" or "too much precision" in science or engineering.
Think of simulating a new passenger jet flying with extended, complicated, wing flaps. Is 2000x2000x2000 resolution sufficient? (Assume the length of aircraft is 70 m.)

A high resolution simulation may need ~1 million time steps to complete. (That's because the computational cells are small and modeled physical signals propagating through the grid of cells cannot cross more than 1 cell per time step. In practice, a physical disturbance of some sort crossing 50 times a grid of 2000 cells in both directions needs a minimum of 500K steps, in agreement with the 1M estimate).

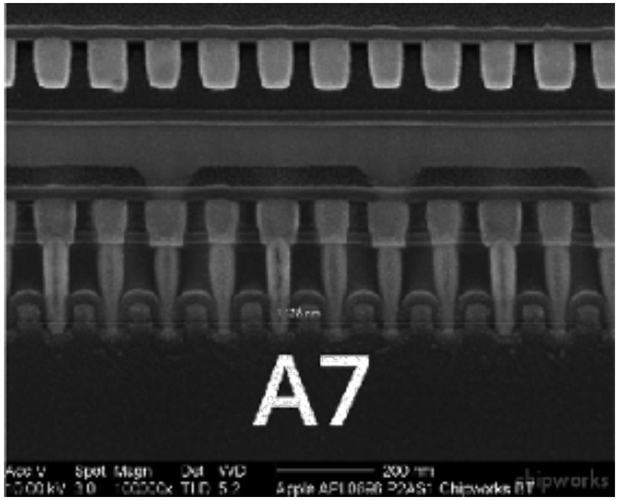
If so, then each of our hypothetical 1000 researchers have to wait a *minimum* of

1M * (0.05s) ~ 14 hours

for their simulations to complete, if everything works at 100% efficiency.
So the seemingly 'ridiculously large' number of arithmetic ops per second (>10^{17}), of which Summit is capable, is not so ridiculous after all when shared among scientists.

- For certain arithmetic-heavy computations, Summit's performance of 150 PFLOPS divided among ~10^3 concurrent jobs is 'only' 150 TFLOPS/job
- Taking into account *bandwidth limitations* (decisive for most computations!), Summit is of course fast but not excessively fast. Neither will be the 1 EFLOP performance. 1 EFLOPS = ExaFLOPS = 10^{18}/s. First such computers will probably emerge in 2021.

Perspectives on future computing


1138 nm
A7
Acc.V Spot Magn Det WD
10.00 kV 3.0 100000x TLD 5.2
200 nm
Apple APL0698 P2AS1 Chipworks BT
chipworks

Apple A7 processor under electron microscope

A7 working in IPhone 5S

Ten transistors are shown, spread over 1138 nm. Technology known as “28 nm” is used in A7 (the width of conductors etched on silicon substrate). But 28 nm is only 70 atomic sizes of silicon.

Since the introduction of A7, manufacturing technology transferred to width 14 & 10 nm, and 7 nm (18 Si atoms!) has begun. But there are tough technological and economic issues, as well as (soon) problems of quantum nature, which will soon get in the way. We may have to drop silicon as substrate and adopt exotic materials such as graphene, or jump into the quantum world directly by producing *quantum computers,* operating on atomic scale & using superposition of quantum states, and so-called qubits instead of bits. Algorithms for them are being developed among others at UTSC.
For quantum computing, today is a pioneer era, like 1940s for von Neumann computers!

PYTHON ver.3 - plan for the next ~2 lectures

Joshua Izaac & Jingbo Wang, “Computational Quant. Mech.”
(Undergrad. Lecture Notes in Physics, Springer, 2018)

chapter 1. Numbers and precision

- 1.1-1.3. Fixed point, Floating point representation of real numbers

chapter 3. Python

- 3.1 Indentation, case, comment
- 3.2 Variables and their precision
- 3.3 Operator precedence, conditional statements
- 3.4 Strings
- 3.5 Data structures
- 3.6 Loops
- 3.7 I/O
- 3.8 Functions
- 3.9 NumPy and arrays
- 3.10 Arguments
- 3.11 Timing the code

-- Differences between Python2 & Python3 (cf. link on course page)

read the book,
come to tutorials!

PYTHON as problem solver, or at least a great helper..

- How to solve $a x^4 + b x^3 + c x^2 + d x + e = 0$?
- How much is ....i^i ?
- How to compute positions of Saturn-class exoplanet K2-261b discovered in 2018, at regular time intervals covering one orbit?
- What maximum distance will a broken physical pendulum land at?
- How likely that a random walk returns to starting point in 2N steps?
- How to compute the minimum, maximum, mean and the standard deviation of a large data set in one pass?
- If a stick is broken at a random point, what's the mean length of the shorter piece & the mean ratio of lengths of two pieces?
- Should I bet that someone draws his/her own ID at random from a hat?
- How to compute oscillation modes of a mechanical system?
- Given a vertical profile of a bike trip, compute total rise & total descent?
- What's the probability that two random points on a unit interval are at distance less than one-half?
- How to fit a polynomial to data? How to fit multi-parameter functions?
- How to compute trajectories of chase, and those in Newton's dynamics?
- Why and how to interpolate by splines?
- Can one compute and animate waves on the surface of water?
- How to simulate realistic airfoil's lift force by attached vortex method?

etc.